

# **Exhibit 4**

**DOCUMENT SOUGHT TO  
BE SEALED**

IN THE UNITED STATES DISTRICT COURT  
FOR THE NORTHERN DISTRICT OF CALIFORNIA  
SAN FRANCISCO DIVISION

|                       |   |                          |
|-----------------------|---|--------------------------|
| ORACLE AMERICA, INC., | ) |                          |
|                       | ) |                          |
| Plaintiff,            | ) |                          |
|                       | ) |                          |
| v.                    | ) | Civ. A. No. 10-03561 WHA |
|                       | ) |                          |
| GOOGLE INC.,          | ) | (Jury)                   |
|                       | ) |                          |
| Defendant.            | ) |                          |

**EXPERT REPORT OF PROFESSOR DOUGLAS C. SCHMIDT, Ph.D. REGARDING FAIR  
USE AND REBUTTAL TO GOOGLE'S OPENING EXPERT REPORTS**

**February 8, 2016**

**HIGHLY CONFIDENTIAL**

224. While I disagree with this conclusion about the absolute necessity of *all* of java.lang to use the language, this position makes clear that Google concedes the remaining 36 Java API packages at issue are not required to use the language, a point the parties can agree on.

225. Dr. Astrachan also references Trial Exhibit 1062 describing the list of 61 classes as interfaces essential to Java in his report. I do agree with Google and Dr. Astrachan that those classes and interfaces outside of the 61 mentioned in Trial Exhibit 1062 are **not** required to program in the Java programming language.

226. Trial Exhibit 1062 was created by Chief Java Architect Mark Reinhold. The 61 classes and interfaces listed in the document bear the distinction of being mentioned by the Java Language Specification outside of an example. As Dr. Reinhold testified, these classes and interfaces are only mentioned but not *specified*.<sup>166</sup> As such they are only required by the Java programming language in a formalistic sense; there is no requirement that these classes and interfaces declare particular methods. There is certainly no requirement that these classes and interfaces have the same declaring code and SSO as exists in the 37 Java API packages. As such, it is my opinion that these 61 classes and interfaces, as expressed in the 37 Java API packages, are not part of or essential to the Java programming language.

227. In summary, it is my opinion that the primary motivation behind Android's infringement of the 37 Java API packages was not the necessity of copying essential classes, or to duplicate method declarations essential to the Java language. Rather, Android Java copied ten times more classes than were strictly necessary.

228. In (¶158-161) Dr. Astrachan lauds the Java platform and describes the numerous creative steps that were taken to promote Java as a programming language. It is my opinion that Android's decision to use the 37 Java API packages was driven by the need to attract existing Java programmers to the Android platform, thereby leveraging the ecosystem that Sun/Oracle had painstakingly developed and supported over the years.

---

<sup>166</sup> Reinhold Trial Tr., 677:3-678:13.

240. However, it is my opinion that not all of the copied declarations represent functional requirements of the language. Instead, Android appropriated content in excess of the absolute minimum amount necessary, and subsequently made this material a critical part of its own platform.

241. To this end, I performed a series of tests involving the build process for the "Lollipop" release of the Android operating system. The tools and testing environment used to perform these tests are identical to the ones used for the build tests discussed in my initial report.<sup>170</sup>

242. I performed the build of this source code under two different scenarios, and logged the resulting output of each build process. A description of each build test scenario is provided below, as well as the modifications to the source code that were performed in order to create the build environment.<sup>171</sup>

243. In the first scenario, I manually removed all of the 37 copied packages, except for the java.lang package, prior to building. Note that source files for packages under java.lang, such as java.lang.ref, were also deleted. Attempting to build Android under this scenario resulted in failure. The log of this build test is found in Appendix O.

244. In the second scenario, I deleted all of the 37 copied packages except for the 61 classes that are mentioned in the third edition of the Java Language Specification (JLS).<sup>172</sup> I searched for these 61 source files and delete them. This build attempt also resulted in failure - the log of the build test is found in Appendix P.

245. Based on the results of these build tests, it is my opinion that Dr. Astrachan's assertions regarding which classes are essential to the language are flawed, as is evidenced by the fact

---

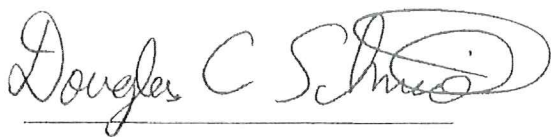
<sup>170</sup> Specifically, this analysis uses Android Lollipop version 5.1.1, release 30. The source code was first downloaded from the online repository on December 17, 2015, using the source code tag 'android-5.1.1\_r30' (LMY48Z). See "Codenames, Tags, and Build Numbers," Android Open Source Project, <https://source.android.com/source/build-numbers.html> (accessed Feb 8, 2016), for the complete list of source code tags.

<sup>171</sup> All source files associated with the 37 API packages are found under the 'libcore' folder of the Android source code. Under this folder, the files are split between the 'libart' and 'luni' subfolders. When performing the source code modifications described above, both of these folders were examined to search for and delete the files of interest to set up the build environment for each scenario.

<sup>172</sup> See TX 1062 for the complete listing of the classes that are mentioned in the third edition of the Java Language Specification.

**XI. ATTESTATIONS**

311. I declare under penalty of perjury under the laws of the United States of America that the foregoing is true and correct. Executed on this 8<sup>th</sup> day of February, 2016, in New York, NY.

A handwritten signature in cursive script, reading "Douglas C Schmidt", written over a horizontal line.

Douglas C. Schmidt, Ph.D.



## IX. APPENDIX H: ACADEMIC LITERATURE REVIEW FOR VISUALIZATION OF THE SSO USING D3

8. D3 makes use of the widely implemented Scalable Vector Graphics ("SVG"), HTML5, and Cascading Style Sheets ("CSS") standards. Embedded within an HTML webpage, the JavaScript D3.js library uses pre-built JavaScript functions to select elements, create SVG objects, style them, or to add transitions, dynamic effects or tooltips. These objects can also be widely styled using CSS. Large datasets can be easily bound to SVG objects using simple D3.js functions to generate rich text/graphic charts and diagrams. Data can be aggregated in various formats, most commonly JavaScript Object Notation ("JSON"), comma-separated values ("CSV") or geoJSON.<sup>[1]</sup>
  
9. As one of the most popular data visualization tools, D3 has been widely used in academic research for various research topics including software development, genetics, biotech, social network, stock, and many others. Due to D3's ability to visualize large scale datasets with great flexibility, it has been widely used in network visualizations with large datasets. Bostock et al's paper on D3 data-driven documents<sup>[2]</sup> has been formally cited over 900 times in published, peer-reviewed scholarly literature. Gouveia et al<sup>[3]</sup> proposed three dynamic graphical forms using D3 to display the diagnostic reports yielded by spectrum-based software fault localization. Tan et al<sup>[4]</sup> developed Network2Canvas, a web application implemented in D3 for large scale network visualization. Yoon et al<sup>[5]</sup> used "AZURITE", which is implemented in JavaScript and D3, to visualize the complex software change history. D3's strong capability in network visualization and dependency visualization makes it an ideal tool to visualize software SSO with large amounts of nodes and edges.

---

<sup>[1]</sup> "Data Driven Documents," <https://d3js.org/> (Accessed Feb. 8, 2016)

<sup>[2]</sup> "D3: Data-Driven Documents," M. Bostock, V. Ogievetsky, and J. Heer, IEEE Transactions on Visualization and Computer Graphics, vol. 17, no. 12, pp. 2301–2309, Dec. 2011. Available at: <http://vis.stanford.edu/files/2011-D3-InfoVis.pdf> (Accessed Feb. 8, 2016)

<sup>[3]</sup> "Using HTML5 Visualizations in Software Fault Localization," C. Gouveia, J. Campos, R. Abreu. (Accessed Feb. 8, 2016)

<sup>[4]</sup> "Network2Canvas: Network Visualization on a Canvas with Enrichment Analysis," C. M. Tan et al. (Accessed Feb. 8, 2016)

<sup>[5]</sup> "Visualization of Fine-Grained Code Change History," Y.S. Yoon et al, Carnegie Mellon University, 2013, [http://www.cs.cmu.edu/~NatProg/papers/P1\\_PP20\\_Yoon%20Azurite%20VLHCC13.pdf](http://www.cs.cmu.edu/~NatProg/papers/P1_PP20_Yoon%20Azurite%20VLHCC13.pdf) (Accessed Feb. 8, 2016)